ROUND ROBIN SCHEDULING ALGORITHM WITH DYNAMIC TIME QUANTUM

By

Aakriti Upadhyay

Masters in Computer Science,

Department of Computer Science

TABLE OF CONTENTS

ABSTRACT

The Round Robin (RR) algorithm is one of the most widely used scheduling algorithms. While it is optimal for time-sharing systems, the static nature of the time quantum can lead to performance issues. A large-time quantum results in low context switching, while a small-time quantum causes high context switching. Consequently, the fixed time quantum can decrease the performance of the round-robin algorithm. This work proposes a dynamic time quantum variance to address the issue. This approach allows for the appropriate calculation of the time quantum for each process as it enters the scheduling queue, leading to better process scheduling and improved multitasking.

The primary objective of this project is to implement a real-time scheduling algorithm that minimizes the average waiting time, ensuring that a given set of tasks is completed in minimal time with efficient output, thanks to the introduction of the dynamic time quantum. In real-time systems, created tasks deliver specific services upon execution, making each task crucial to the system's overall functionality. Unlike non-real-time systems, which do not consider deadlines, real-time systems prioritize deadlines when scheduling tasks.

PROPOSED ALGORITHM

The processes in the ready queue are arranged in the ascending order of their remaining burst time. CPU is allocated to the processes using RR scheduling with time quantum value equal to the mean of burst time of all arrived processes in the ready queue. After each cycle processes in the ready queue are arranged in the ascending order of their remaining burst time and CPU is allocated to the processes using RR scheduling with time quantum value equal to the mean of burst time of remaining processes in the ready queue. The steps are as follows:

Step 1: Arrange the processes in the increasing order of CPU burst time in the ready queue.

Step 2: Calculate the Mean of the CPU burst time of all the Processes Mean (M) = (P1+P2+P3……….Pn)/n;

Step 3: Set the dynamic quantum time (DQT) per the following method DQT = M;

Step 4: Allocate all processes to the CPU like present Round Robin scheduling algorithm in the arranged order as in Step1, with time period of Dynamic Time Quantum.

Step 5: If remaining CPU burst time is less than the DTQ, perform the Step2 and Step3 again.

Step 6: Continue till a new process is inserted into the queues, in case repeat all steps from Step1.

Step 7: Process the queue till all processes are executed.

<u>Flowchart Representation</u> **:**

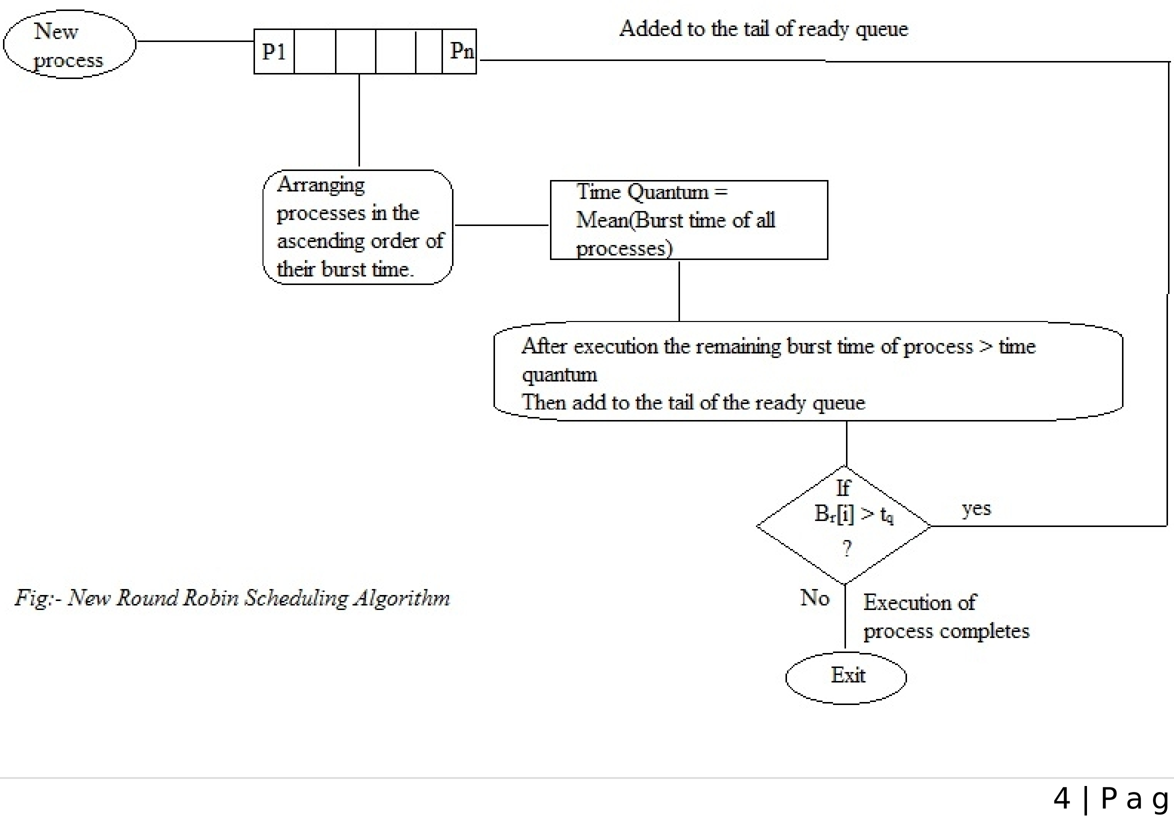


Fig:- New Round Robin Scheduling Algorithm

IMPLEMENTATION CODE

```
#include<stdio.h>

//DISPLAYING THE GANTT CHART
void Gantt_chart(int n,int pname[], int Wt[], int Bu[])
{
    int i;
    printf("\n\nGANTT CHART");
    printf("\n------------------------------------------------------------------\n");
    for(i=0;i<n;i++)
        printf("|\tP%d\t",pname[i]);
    printf("|\t");
    printf("\n------------------------------------------------------------------\n");
    //printf("\n");
    for(i=0;i<n;i++)
        printf("%d\t\t",Wt[i]);
    printf("%d",Wt[n-1]+Bu[n-1]);
    printf("\n------------------------------------------------------------------\n");
    printf("\n");
 }

int main()
{
            int i, limit, total = 0, x, counter = 0, time_quantum, y=0, p[10], t,
t1=0, pos,Wt[10];
            int wait_time = 0, turnaround_time = 0, arrival_time[10],
burst_time[10], temp[10];
            float average_wait_time, average_turnaround_time;
            printf("\nEnter Total Number of Processes:\t");
            scanf("%d", &limit);
            x = limit;
            for(i = 0; i < limit; i++)
            {
                        printf("\nEnter Details of Process[%d]\n", i + 1);
                        printf("Arrival Time:\t");
                        scanf("%d", &arrival_time[i]);
                        printf("Burst Time:\t");
                        scanf("%d", &burst_time[i]);
                //y = y + burst_time[i];
                 p[i]=i+1; //contains process number
            }

            printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting
Time\n");
            for(total = 0, i = 0; x != 0;)
            {
              //sorting burst time in ascending order using selection sort
                  for(int k = i; k < limit; k++)
```

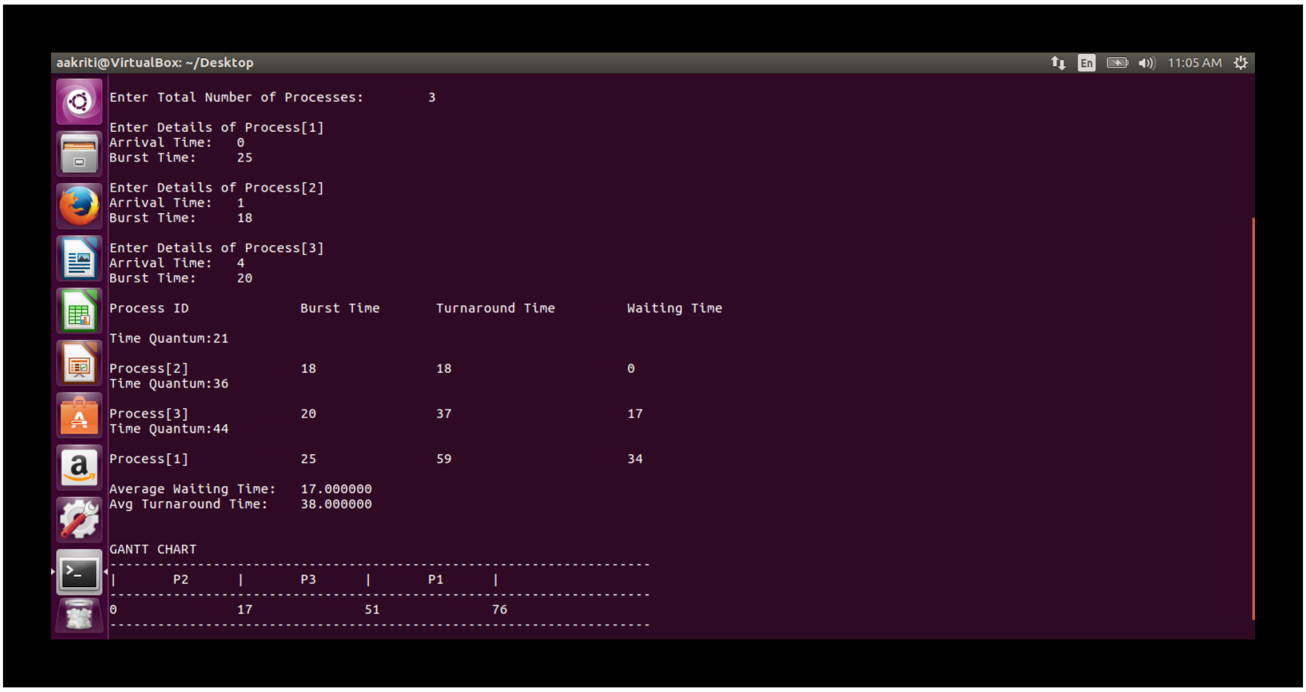
```
                {
                        pos=k;
                        for(int j=k+1; j < limit; j++)
                        {
                          if(burst_time[j]<burst_time[pos])
                              pos=j;
                        }

                        t=burst_time[k];
                        burst_time[k]=burst_time[pos];
                        burst_time[pos]=t;

                        t1=p[k];
                        p[k]=p[pos];
                        p[pos]=t1;
                }

                for(int k = i; k < limit; k++)
                {  temp[k] = burst_time[k];
                   y = y + burst_time[k];
                 }

//Dynamic Time Quantum is the average of burst time of all arrived process
                   time_quantum = y/limit-i;
                printf("\nTime Quantum:%d\t\n", time_quantum);

                if(temp[i] <= time_quantum && temp[i] > 0)
                {
                        total = total + temp[i];
                        temp[i] = 0;
                        counter = 1;
                }
                else if(temp[i] > 0)
                {
                        temp[i] = temp[i] - time_quantum;
                        total = total + time_quantum;
                }
                if(temp[i] == 0 && counter == 1)
                {
                        x--;
                        printf("\nProcess[%d]\t\t%d\t\t   %d\t\t\t   %d",   p[i],
burst_time[i], total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
                        wait_time  =  wait_time  +  total  -  arrival_time[i]  -
burst_time[i];
                Wt[i]=wait_time;
                        turnaround_time   =   turnaround_time   +   total   -
arrival_time[i];
                        counter = 0;
                }
```

```
            if(i == limit - 1)
            {
                i = 0;
            }
            else if(arrival_time[i + 1] <= total)
            {
                i++;
            }
            else
            {
                i = 0;
            }
        }
        average_wait_time = wait_time * 1.0 / limit;
        average_turnaround_time = turnaround_time * 1.0 / limit;
        printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
        printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);
      Gantt_chart(limit, p, Wt, burst_time);
        return 0;
}
```

OUTPUT

```
aakriti@VirtualBox:~/Desktop$ ./a.out

Enter Total Number of Processes:    4

Enter Details of Process[1]
Arrival Time:  0
Burst Time:    18

Enter Details of Process[2]
Arrival Time:  1
Burst Time:    20

Enter Details of Process[3]
Arrival Time:  3
Burst Time:    24

Enter Details of Process[4]
Arrival Time:  4
Burst Time:    60

Process ID            Burst Time      Turnaround Time       Waiting Time

Time Quantum:30

Process[1]            18              18                    0
Time Quantum:34

Process[2]            20              37                    17
Time Quantum:42

Process[3]            24              59                    35
Time Quantum:60

Process[4]            60              118                   58

Average Waiting Time: 27.500000
Avg Turnaround Time:  58.000000


GANTT CHART
--------------------------------------------------------------------
|       P1      |       P2      |       P3      |       P4      |
--------------------------------------------------------------------
0               17              52              110             170
--------------------------------------------------------------------
```

Screenshot :



Graphs :

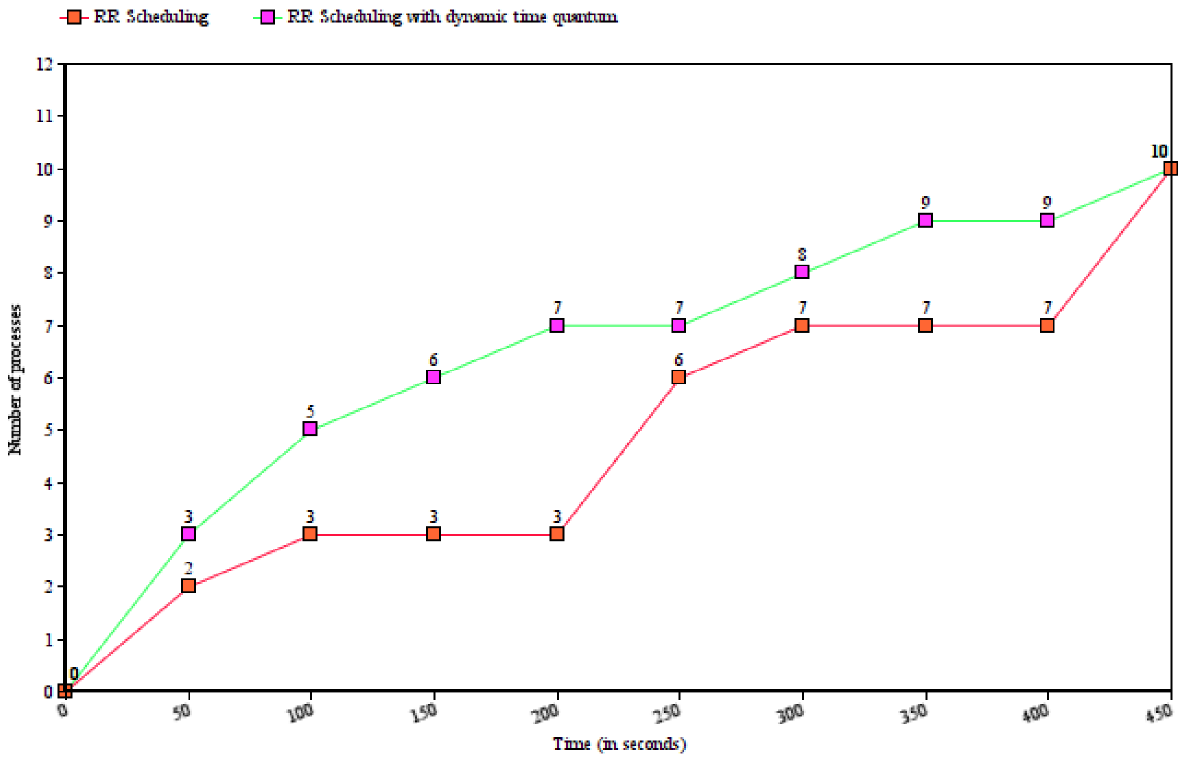
The graphs below compares RR_{old} and RR_{new} scheduling algorithms based on turnaround time, waiting time and average times for 10 processes.

Plot 1 represents the CPU utilisation by each algorithm considering the number of processes executes completely with respect to the turnaround time.

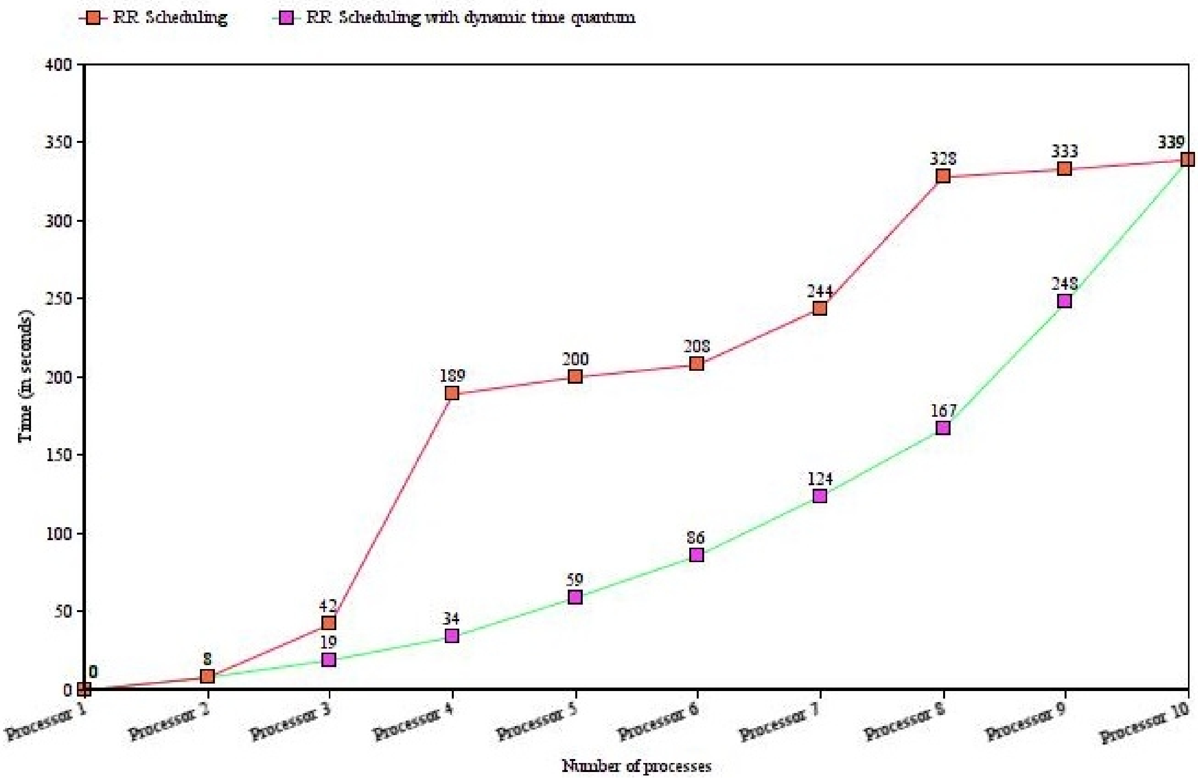
Plot 2 represents the waiting time for each processor before execution to complete for the two algorithms.

Plot 3 compares the average waiting time and average turnaround time of each of the algorithms over 20 runs.

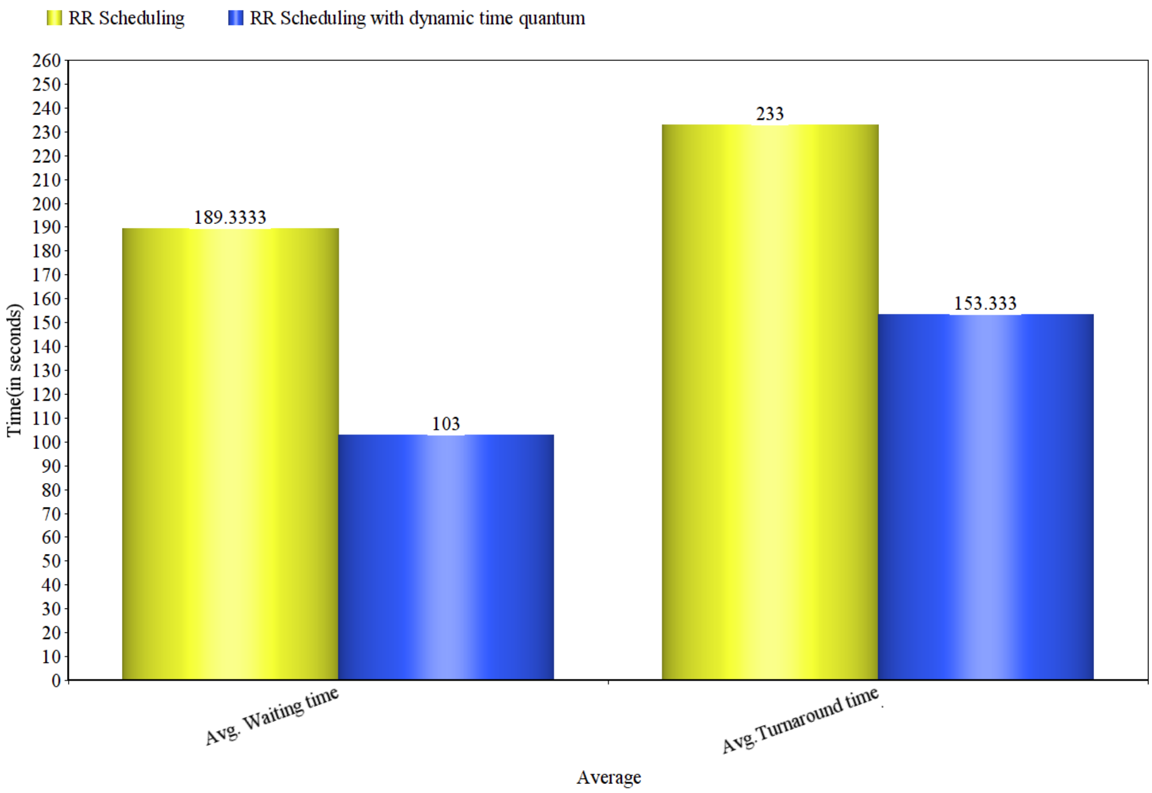
The red line indicates the working of RR_{old} scheduling algorithm and green line indicates the RR_{new} scheduling algorithm.



Plot 1: **CPU Utilisation time (Turnaround Time)**



Plot 2: **Waiting Time of each Processor**



Plot 3: **Performance Analysis**

CONCLUSION

The improved round robin scheduling algorithm with varying quantum time shows better scheduling performance results in the plots compared to the existing round robin scheduling algorithm. In plot 1, the rate of CPU utilization of new algorithm is higher than the old algorithm within a time range. The processes gets executed to complete with minimum waiting time during scheduling by new algorithm. The performance analysis of new algorithm is far better than the old algorithm. This algorithm can be used for CPU scheduling in the future systems.

BIBLIOGRAPHY

[1] Aakriti Upadhyay and Siddheshwar Kanagamoorthy, "*Improved Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum*", Paper 1 of the Operating Systems Class SUNY ALBANY, October 2016.

[2] Aakriti Upadhyay and Siddheshwar Kanagamoorthy, "*Dynamic Time Quantum Estimation for Effective Round Robin CPU Scheduling*", Paper 2 of the Operating Systems Class SUNY ALBANY, November 2016.

[3] Data, Retrieved December 11, 2016, from https://www.google.com/.

[4] Round-robin scheduling, (2016, November 16) retrieved December 12, 2016, from https://en.wikipedia.org/wiki/Round-robin_scheduling.